Games Solutions: OpenPlay with NetSprocket

Session 512

Games Solutions: OpenPlay With NetSprocket

Lane Roathe
President
Ideas From the Deep

Introduction

- What is OpenPlay?
- Discussion of the open source effort
- Brief description of the APIs
- Using the NetSprocket API
- Demo using OpenPlay
- OpenPlay vs. other networking APIs
- Summary and documentation

Introduction

What is OpenPlay?

- An Apple Open Source project
- Protocol and Platform independent APIs
- Consists of three layers
    - Game (NetSprocket)
    - Protocol (OpenPlay)
    - NetModule (AppleTalk, TCP/IP, etc)

Introduction

History of the APIs

- OpenPlay
  - Uber, developed by Bungie
  - Platform/Protocol independent design
- NetSprocket
  - Part of Game Sprockets
  - Originally based on OpenTransport

Today's OpenPlay

- Added NetSprocket APIs
    - NetSprocket now cross platform
    - Backward compatible with 1.7
- Active Open Source Community
- Used in several shipping titles
- Posix version for Linux, Mac OS X, etc.
    - Windows build now based on posix code

Introduction

Open source effort

- Project is actively moving forward
- Recent 2.1 release
- How can I contribute?
    - Visit **www.darwin.org**
    - Submit modifications
    - Help with documentation

OpenPlay

Overview

- Purpose
- Available API layers
    - NetSprocket
    - Protocol
    - NetModule
- API Overviews

OpenPlay

Purpose

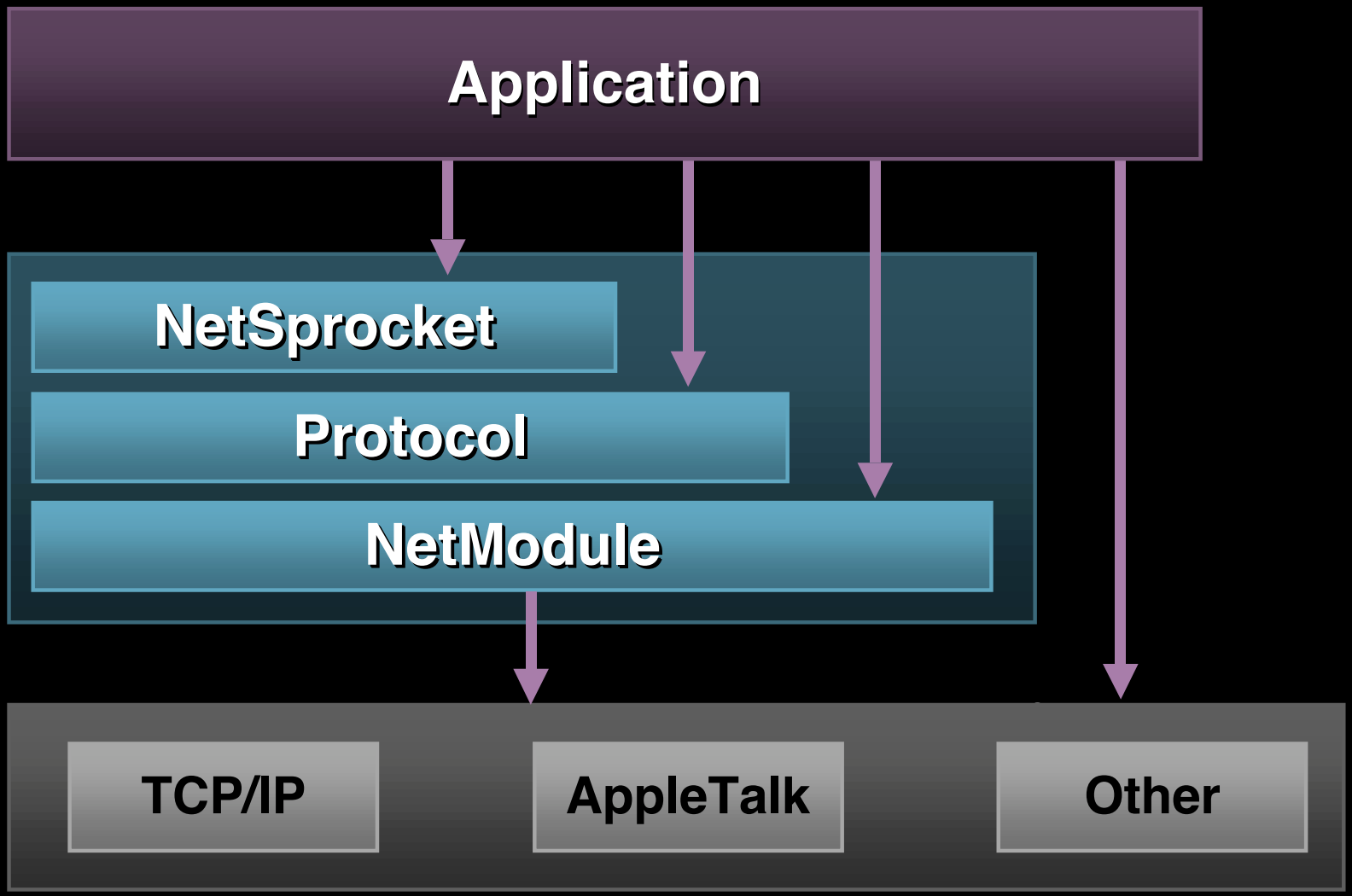
- Ease the effort required to implement networking into games and applications
- Cross Platform Solution
    - Mac OS 8/9 and X, Windows, Linux
- Protocol independence
    - AppleTalk, TCP/IP

OpenPlay

Available API layers

- NetSprocket
- Protocol
- NetModule

OpenPlay


Application
NetSprocket
Protocol
NetModule
TCP/IP
AppleTalk
Other

OpenPlay

NetSprocket

- Manages games, users and groups
- Uses Protocol API for networking
- Very easy to implement and use
- Need only worry about game data
- Handles routing, byte swapping, packet assembly, and other details

OpenPlay

Protocol

- Abstracts networking API
- Uses NetModule API for networking
- Easier to use than raw networking APIs, with more functionality
- Game must manage connections and data, including byte swapping

OpenPlay

NetModule

- Wrapper layer for raw networking APIs
- Networking API specific features
- Mainly for use by Protocol API
- Few games will need to use this API
- End of coverage in this session
    - See Writing_OP_NetModules for more

New Example Code

Facilitates learning and using OpenPlay

- Example new to 2.1 release
- Designed to be useful and easy to follow
- NetSprocket and Protocol API builds
- Fully functional client/server game
- OpenPlay code in separate files
- Can be used as starting point in your app

NetSprocket Usage

Overview of using NetSprocket API in your app

- Covers OpenPlay's NetSprocket API
- Taken from new Example 1 source
- For compiling/working source see archive
- Usage of Protocol and NetModule APIs not covered in this session
    - In general Protocol layout is the same
    - Protocol code usage in op_network.cpp

NetSprocket Example

- Startup

```
// Make sure NetSprockets is available
If ( NULL == NSpInitalize )
      return( errModuleNotFound );

// Init NetSprocket to default values
// ‘NSe1’ is our unique ID
err = NSpInitialize( 0, 0, 0, ‘NSe1’, 0 );
```

NetSprocket Example

- Startup

```
// Make sure NetSprockets is available
If ( NULL == NSpInitalize )
      return( errModuleNotFound );

// Init NetSprocket to default values
// ‘NSe1’ is our unique ID
err = NSpInitialize( 0, 0, 0, ‘NSe1’, 0 );
```

NetSprocket Example

- Start Server

```
// First, we need to create a protocol list and add
// an IP reference to that list
NSpPrococolListRefernence plRef = NULL;
NSpProtocolReference pRef = NULL;

Err = NSpProtocolList_New( NULL, &plRef );
If ( !Err )
     pRef = NSpProtocol_CreateIP( port, 0, 0 );
If ( pRef )
     Err = NSpProtocolList_Append( plRef, pRef );
```

NetSprocket Example

- Start Server

```
// First, we need to create a protocol list and add
// an IP reference to that list
NSpPrococolListRefernence plRef = NULL;
NSpProtocolReference pRef = NULL;

Err = NSpProtocolList_New( NULL, &plRef );
If ( !Err )
     pRef = NSpProtocol_CreateIP( port, 0, 0 );
If ( pRef )
     Err = NSpProtocolList_Append( plRef, pRef );
```

NetSprocket Example

- Start Server

```
// First, we need to create a protocol list and add
// an IP reference to that list
NSpPrococolListRefernence plRef = NULL;
NSpProtocolReference pRef = NULL;

Err = NSpProtocolList_New( NULL, &plRef );
If ( !Err )
     pRef = NSpProtocol_CreateIP( port, 0, 0 );
If ( pRef )
     Err = NSpProtocolList_Append( plRef, pRef );
```

NetSprocket Example

- Start Server

```
// Got everything setup, now we can host
NSpGameReference gameRef = NULL;

Err = NSpGame_Host( &gameRef, plRef,
      maxPlayers, gameName, gamePassword,
      kPlayerType, kNSpClientServer, 0 );

// Game is now listening on spec. port, and will
// send a message when a client connects.
```

NetSprocket Example

- Start Server

```
// Got everything setup, now we can host
NSpGameReference gameRef = NULL;

Err = NSpGame_Host( &gameRef, plRef,
      maxPlayers, gameName, gamePassword,
      kPlayerType, kNSpClientServer, 0 );

// Game is now listening on spec. port, and will
// send a message when a client connects.
```

NetSprocket Example

- Start Server

```
// Got everything setup, now we can host
NSpGameReference gameRef = NULL;

Err = NSpGame_Host( &gameRef, plRef,
      maxPlayers, gameName, gamePassword,
      kPlayerType, kNSpClientServer, 0 );

// Game is now listening on spec. port, and will
// send a message when a client connects.
```

NetSprocket Example

- Start Server

```
// Got everything setup, now we can host
NSpGameReference gameRef = NULL;

Err = NSpGame_Host( &gameRef, plRef,
      maxPlayers, gameName, gamePassword,
      kPlayerType, kNSpClientServer, 0 );

// Game is now listening on spec. port, and will
// send a message when a client connects.
```

NetSprocket Example

- Start Server

```
// Got everything setup, now we can host
NSpGameReference gameRef = NULL;

Err = NSpGame_Host( &gameRef, plRef,
      maxPlayers, gameName, gamePassword,
      kPlayerType, kNSpClientServer, 0 );

// Game is now listening on spec. port, and will
// send a message when a client connects.
```

NetSprocket Example

- Start Server

```
// Got everything setup, now we can host
NSpGameReference gameRef = NULL;

Err = NSpGame_Host( &gameRef, plRef,
      maxPlayers, gameName, gamePassword,
      kPlayerType, kNSpClientServer, 0 );

// Game is now listening on spec. port, and will
// send a message when a client connects.
```

NetSprocket Example

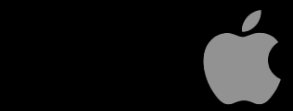
- Start Server

```
// Got everything setup, now we can host
NSpGameReference gameRef = NULL;

Err = NSpGame_Host( &gameRef, plRef,
      maxPlayers, gameName, gamePassword,
      kPlayerType, kNSpClientServer, 0 );

// Game is now listening on spec. port, and will
// send a message when a client connects.
```

NetSprocket Example

- Start Client

```
// First create an IP address ref from IP and port
NSpAddressReference addrRef = NULL;

addrRef = NSpCreateAddressReference (
                              address, port// C Strings
                    )
// Then start the join procedure
If ( addrRef )
      err = NSpGame_Join( &gameRef, addrRef,
                    playerName, password, kPlayerType,
                    NULL, 0, 0 );
```



NetSprocket Example

- Start Client

```
// First create an IP address ref from IP and port
NSpAddressReference addrRef = NULL;

addrRef = NSpCreateAddressReference(
                                address, port)

// Then start the join procedure
If ( addrRef )
     err = NSpGame_Join( &gameRef, addrRef,
                    playerName, password, kPlayerType,
                    NULL, 0, 0 );
```

NetSprocket Example

- Start Client

```
// First create an IP address ref from IP and port
NSpAddressReference addrRef = NULL;

addrRef = NSpCreateAddressReference(
                                address, port)

// Then start the join procedure
If ( addrRef )
      err = NSpGame_Join( &gameRef, addrRef,
                  playerName, password, kPlayerType,
                  NULL, 0, 0 );
```

NetSprocket Example

- Start Client

```
// First create an IP address ref from IP and port
NSpAddressReference addrRef = NULL;

addrRef = NSpCreateAddressReference(
                                address, port)

// Then start the join procedure
If ( addrRef )
      err = NSpGame_Join( &gameRef, addrRef,
                  playerName, password, kPlayerType,
                  NULL, 0, 0 );
```

NetSprocket Example

- Start Client

```
// First create an IP address ref from IP and port
NSpAddressReference addrRef = NULL;

addrRef = NSpCreateAddressReference(
                                address, port)

// Then start the join procedure
If ( addrRef )
      err = NSpGame_Join( &gameRef, addrRef,
                  playerName, password, kPlayerType,
                  NULL, 0, 0 );
```

NetSprocket Example

- Start Client

```
// First create an IP address ref from IP and port
NSpAddressReference addrRef = NULL;

addrRef = NSpCreateAddressReference(
                                address, port)

// Then start the join procedure
If ( addrRef )
      err = NSpGame_Join( &gameRef, addrRef,
                  playerName, password, kPlayerType,
                  NULL, 0, 0 );
```

NetSprocket Example

- Start Client

```
// First create an IP address ref from IP and port
NSpAddressReference addrRef = NULL;

addrRef = NSpCreateAddressReference(
                                address, port)

// Then start the join procedure
If ( addrRef )
      err = NSpGame_Join( &gameRef, addrRef,
                  playerName, password, kPlayerType,
                  NULL, 0, 0 );
```

NetSprocket Example

- Start Client

```
// Now loop for a while waiting for accept/deny
while( !response && currentTime < expirationTime)
{
    NSpMessageHeader *mhp;

    mhp = NSpMessage_Get( gameRef );
    if ( mhp )
        switch( mhp->what )
}
```

NetSprocket Example

- Start Client

```
// Now loop for a while waiting for accept/deny
while( !response && currentTime < expirationTime)
{
    NSpMessageHeader *mhp;

    mhp = NSpMessage_Get( gameRef );
    if ( mhp )
        switch( mhp->what )
}
```

NetSprocket Example

- Start Client

```
// Host approved your join request, we're ready to go

case kNSpJoinApproved:
    response = true;
    approved = true;
    break;
```

NetSprocket Example

- Start Client

```
// Host rejected your join request, or there was some
// type of error that prevented you from joining.

case kNSpJoinDenied:      // most common reason
case kNSpGameTerminate:
case kNSpError:           // error or game ended
    response = true;
    approved = false;
    break;
```

NetSprocket Example

- Start Client

```
// Once we have handle the message, we need to
// dispose of the message to free it's memory

NSpMessage_Release ( gameRef, mhp );

// To save code, you can handle these messages
// in the main NSp message handling code, which
// is what the Example1 code does.
```

NetSprocket Example

- Send Message

```
// Struture used to send a game message

typedef struct
{
    NSpMessageHeader header;

    char    message[1];

} tMessagePacket, MessagePacketPtr;
```

NetSprocket Example

- Send Message

```
// Struture used to send a game message

typedef struct
{
    NSpMessageHeader header;

    char    message[1];

} tMessagePacket, MessagePacketPtr;
```

NetSprocket Example

- Send Message

```
// Get buffer to hold message

MessagePacketPtr mpp;

// assume c string passed in as message
msgLen = strlen( message );
// size of our buffer is header, len, and null term
size = sizeof(tMessagepacket) + msgLen + 1;
// allocate buffer
mpp = (MessagePacketPtr) malloc( size );
```

NetSprocket Example

- Send Message

```
// Get buffer to hold message

MessagePacketPtr mpp;

// assume c string passed in as message
msgLen = strlen( message );
// size of our buffer is header, len, and null term
size = sizeof(tMessagepacket) + msgLen + 1;
// allocate buffer
mpp = (MessagePacketPtr) malloc( size );
```

NetSprocket Example

- Send Message

```
// Get buffer to hold message

MessagePacketPtr mpp;

// assume c string passed in as message
msgLen = strlen( message );
// size of our buffer is header, len, and null term
size = sizeof(tMessagepacket) + msgLen + 1;
// allocate buffer
mpp = (MessagePacketPtr) malloc( size );
```

NetSprocket Example

- Send Message

```
// Get message ready to send

NSpClearMessageHeader( &mpp->header );

mpp->header.what = messageType;
mpp->header.to = kNSpAllPlayers;
mpp->header.messageLen = size;

strcpy( mpp->message, message );
```

NetSprocket Example

- Send Message

```
// Get message ready to send

NSpClearMessageHeader( &mpp->header );

mpp->header.what = messageType;
mpp->header.to = kNSpAllPlayers;
mpp->header.messageLen = size;

strcpy( mpp->message, message );
```

NetSprocket Example

- Send Message

```
// Get message ready to send

NSpClearMessageHeader( &mpp->header );

mpp->header.what = messageType;
mpp->header.to = kNSpAllPlayers;
mpp->header.messageLen = size;

strcpy( mpp->message, message );
```

NetSprocket Example

- Send Message

```
// Get message ready to send

NSpClearMessageHeader( &mpp->header );

mpp->header.what = messageType;
mpp->header.to = kNSpAllPlayers;
mpp->header.messageLen = size;

strcpy( mpp->message, message );
```

NetSprocket Example

- Send Message

```
// Get message ready to send

NSpClearMessageHeader( &mpp->header );

mpp->header.what = messageType;
mpp->header.to = kNSpAllPlayers;
mpp->header.messageLen = size;

strcpy( mpp->message, message );
```

NetSprocket Example

- Send Message

```
// Finally, we can have NetSprocket send mesage

Err = NSpMessage_Send ( gameRef,
                        &mpp->header,
                        kNSpSendFlag_Registered
                      );
```

NetSprocket Example

- Send Message

```
// Finally, we can have NetSprocket send mesage

Err = NSpMessage_Send ( gameRef,
                        &mpp->header,
                        kNSpSendFlag_Registered
                      );
```

NetSprocket Example

- Send Message

```
// Finally, we can have NetSprocket send mesage

Err = NSpMessage_Send ( gameRef,
                        &mpp->header,
                        kNSpSendFlag_Registered
                      );
```

NetSprocket Example

- Send Message

```
// Finally, we can have NetSprocket send mesage

Err = NSpMessage_Send ( gameRef,
                        &mpp->header,
                        kNSpSendFlag_Registered
                      );

// _Registered = guarenteed (stream)
// _Normal = one try (datagram)
// _Junk = one try, when nothing else is happening
```

NetSprocket Example

- Send Message

```
// Finally, we can have NetSprocket send mesage

Err = NSpMessage_Send ( gameRef,
                        &mpp->header,
                        kNSpSendFlag_Registered
                      );

// _Registered = guarenteed (stream)
// _Normal = one try (datagram)
// _Junk = one try, when nothing else is happening
```

NetSprocket Example

- Send Message

```
// Finally, we can have NetSprocket send mesage

Err = NSpMessage_Send ( gameRef,
                        &mpp->header,
                        kNSpSendFlag_Registered
                      );

// _Registered = guarenteed (stream)
// _Normal = one try (datagram)
// _Junk = one try, when nothing else is happening
```

NetSprocket Example

- Send Message

```
// Don’t forget to free the temporary storage
    free( mhp );
```

NetSprocket Example

- Get Player Count

```
// Example of how to get game information
NSpGameInfo gameInfo;

Err = NSpGame_GetInfo( gameRef, &gameInfo );
if ( !err )
    return( gameInfo.curentPlayers );
else
    return( 0 );
```

NetSprocket Example

- Get Player Count

```
// Example of how to get game information
NSpGameInfo gameInfo;

Err = NSpGame_GetInfo( gameRef, &gameInfo );
if ( !err )
    return( gameInfo.curentPlayers );
else
    return( 0 );
```

NetSprocket Example

- Handling Messages

```
NSMessageHeader *mhp;

mhp = NSpMesage_Get( gameRef );
if( mhp )
{
    switch( mhp->what )
}
```

NetSprocket Example

- Handling Messages

```
NSMessageHeader *mhp;

mhp = NSpMesage_Get( gameRef );
if( mhp )
{
    switch( mhp->what )
}
```

NetSprocket Example

- Handling Messages

```
// example list of incoming messages types

    // NetSprocket specific
    case kNSpPlayerJoined:
    case kNSpPlayerLeft:

    // Application defined
    case kMessageType_Message:
    case kMessageType_Question:
    case kMessageType_Answer:
```

NetSprocket Example

- Handling Messages

```
// partial list of incoming messages types

    // NetSprocket specific
    case kNSpPlayerJoined:
    case kNSpPlayerLeft:

    // Application defined
    case kMessageType_Message:
    case kMessageType_Question:
    case kMessageType_Answer:
```

NetSprocket Example

- Handling Messages

```
// Once we have handle the message, we need to
// dispose of the message to free it's memory

NSpMessage_Release ( gameRef, mhp );
```

NetSprocket Example

- Shutdown

```
// First, if we have a game active, tell NetSprocket to
// dispose of it, which will also cause the appropriate
// messages to be sent to the host and/or clients.

If ( gameRef )
    NSpGame_Dispose( gameRef,
                     kNSpGameFlag_ForceTermination
                   );
```

NetSprocket Example

- Shutdown

```
// First, if we have a game active, tell NetSprocket to
// dispose of it, which will also cause the appropriate
// messages to be sent to the host and/or clients.

If ( gameRef )
     NSpGame_Dispose( gameRef,
                      kNSpGameFlag_ForceTermination
                    );

// Client is done (ie, there is not an uninitialize call)
```

NetSprocket Example

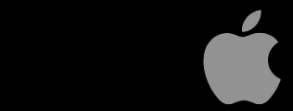
- Shutdown Server

```
// Next we need to get rid of any list stuff we used
// if we were hosting this game.

If ( protocolListRef )
{
    NSpProtocolReference pRef;

    refCount = NSpProtocolList_GetCount(
                                    protocolListRef
                                );
```

NetSprocket Example

- Shutdown Server

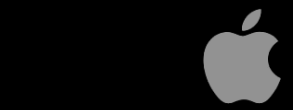
```
// Loop thru list and dispose of refs.

while ( refCount--- && ! err )
{
    pRef = NSpProtocolList_GetIndexRef(
                                        protocolListRef,
                                        refCount );

    err = NSpProtocolList_RemoveIndexed(
                                         protocolListRef,
                                         refCount );
```



NetSprocket Example

- Shutdown Server

```
// Loop thru list and dispose of refs.

while ( refCount--- && ! err )
{
    pRef = NSpProtocolList_GetIndexRef(
                                        protocolListRef,
                                        refCount );

    err = NSpProtocolList_RemoveIndexed(
                                        protocolListRef,
                                        refCount );
```



NetSprocket Example

- Shutdown Server

```
        // Dispose of the actual reference.

        NSpProtocol_Dispose( pRef );
    }

    // After list is emptied and references disposed
    // we can finally dispose of the list

    NSpProtocolList_Dispose( protocolListRef );
```

NetSprocket Example

- Shutdown Server

```
        // Dispose of the actual reference.

        NSpProtocol_Dispose( pRef );
    }

    // After list is emptied and references disposed
    // we can finally dispose of the list

    NSpProtocolList_Dispose( protocolListRef );
```

Protocol Usage

Overview

- Requires more code than NSp
- Complexity issues to deal with
- Can follow same basic outline
- Following slides cover generalities
- See example code for details

Protocol Usage

Startup

- For Mac OS only, must startup OT
    - Except for Mac OS X posix build

Protocol Usage

Startup Server

- Create a protocol config
- Retrieve the assoc. config string
- Replace the port (for listening on)
- Create a new configuration
- Open an Endpoint using 2nd config
- Start a client connection for host if your game has the host play as well

Protocol Usage

Start Client

- Create a protocol config
- Retrieve the assoc. config string
- Replace port and address entries
    - Example program has a replace function
- Create a new protocol config
- Open an endpoint to server

Protocol Usage

Send Messages

- Create Header (which you define)
- Fill in data to send
- Make sure header data in correct byte order (standard is big endian)
- Use ProtocolSendPacket to send
- Must loop thru destinations manually, meaning you must track users, etc.

Protocol Usage

Received Messages

- Messages are received via an interrupt driven callback function
- How to store and retrieve data in main application is up to you
- Example program uses a linked list which requires interrupt safe memory

Protocol Usage

Callback Function

- Handle callback codes in OpenPlay.h
- Must reassemble stream packets which may arrive smaller or larger than sent
- Use ProtocolReceivePacket function to get data from OpenPlay buffers
- Remember to handle byte swapping
- Do not handle messages if possible, queue them up for the main loop

Protocol Usage

Shutdown

- Close all open protocol endpoints
  - Server must close client connections including to self if a host/play game
  - Clients close server and any direct connections it may open
- Wait for all endpoints to finish closing
  - While loop handling messages
- Mac OS 8/9/Carbon must close OT

Protocol Usage

Summary

- With more control comes complexity
- Need to be very careful with the callback function and it is related to interrupt issues
- Need an interrupt safe memory system
- Example code provides all of this, and makes a good starting point. It will need more modifications for your game over the NSp example code

Shipping Titles

Partial list of third-party products

3D Bridge Deluxe
Age of Empires II
Alpha Centauri
Alien Crossfire
AstroRock 2000
Burning Monkey Puzzle Lab
Classic Cribbage
Colin's Classic Cards
Combat Mission: Beyond Overlord
Combat Mission: Barbarossa to Berlin
Galactica: Anno Dominari
Gamesmith
Links Championship Edition
Links LS 2000
Star Wars, Galactic Battlegrounds
Stronghold
Risk II
Total Annihilation
Tony Hawk Pro Skater 2
Oberin
Colin's Bridge
Scrabble
Total Annihilation
Battle for the Universe
Myth, the Fallen Lords
Myth II, Soulblighter
Myth III, the Wolf Age
Last Contact
TacOps - U.S. Army release
TacOpsCav

Demo

GameSmith and NetSprocket Example

API Comparison

Overview

- DirectPlay
- OpenTransport and Sockets

API Comparison

DirectPlay

- Similarities
- Pros/Cons
- Porting hints

API Comparison

DirectPlay

- Similarities
  - Compares to NetSprocket
  - High-level API
  - Handle games, players, and groups
  - Message type flexibility
  - Message reassembly

API Comparison

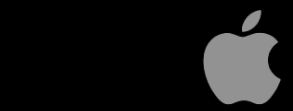
DirectPlay

- Pros to using OpenPlay
  - Platform independent (byte swapping)
  - Ease of use
  - Less vulnerable to system upgrades
  - Low- and high-level APIs
  - Availability of source code
    - Enables you to add features or fix bugs
    - Allows for support of future systems

API Comparison

DirectPlay

- Cons to using OpenPlay
  - User interface not as robust
  - Lacking the voice functionality
  - Not as full featured (traffic, stats)
  - Smaller development community
  - Documentation is lacking



API Comparison

DirectPlay

- Porting hints
  - Use NetSprocket API's
  - Look at the new example project code, specifically nsp_network.c
  - Factor out your networking code
  - Re-implement Windows version, since OpenPlay and DirectPlay do not talk

API Comparison

OpenTransport, Sockets, etc.

- Similarities
- Pros/Cons
- Porting hints

API Comparison

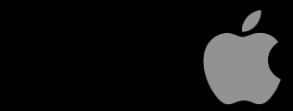
OpenTransport, Sockets, etc.

- Similarities
  - Compares to OpenPlay's Protocol API
  - Stream and datagram connections
  - TCP/IP transport
  - OpenPlay is a wrapper for these APIs

API Comparison

OpenTransport, Sockets, etc.

- Pros to using OpenPlay
  - Provides easier to use APIs
  - Higher level NSp API saves effort and time
  - Much of the detail work is done for you
  - Cross platform
  - Unique features and protocol independence
  - Enumeration ability for game finding
  - Availability of source code



API Comparison

OpenTransport, Sockets, etc.

- Cons to using OpenPlay
  - Loose full control of transport layer
  - Adds a small amount of overhead
  - Might be overkill for simple uses
  - OpenPlay not fully optimized
  - Documentation is lacking

API Comparison

OpenTransport, Sockets, etc.

- Porting hints
  - Look at the new example project
  - Factor out the networking code
  - Use the Protocol APIs for similarity
  - Using the NetSprocket APIs may prove easier even if it means a new networking implementation

Summary

- OpenPlay and NetSprocket have merged
- Makes networking easy
- Choice of high and low level APIs
- Cross platform (Mac OS, Windows, linux)
- Protocol Independent (Appletalk, TCP/IP)
- Open Source with active members
- Contributions welcome!

Technical Documentation

- In the Documentation folder
  - **NetSprocket 1.7**—a little out of date, see OpenPlay.h and Testing.rtf after reviewing
  - **OpenPlay API**—Good general description of Protocol API
  - **OP_Programmers_Docs**—Older Protocol API reference
  - **Writing_OP_NetModules**—Description of using and writing NetModules

Roadmap

100 Darwin Roadmap	Room A1 **Mon., 2:00pm**
803 Mac OS X Networking Overview	Room A2 **Tue., 9:00am**
103 Open Source, Apple, and You	Civic **Tue., 2:00pm**
504 OpenGL: Graphics Programmability	Room A2 **Tue., 5:00pm**

Roadmap

505 OpenGL: Integrated Graphics 1	Room J **Wed., 9:00am**
506 OpenGL: Integrated Graphics 2	Room J **Wed., 10:30am**
FF002: Darwin	Room J1 **Wed., 3:30pm**
112 Writing Threaded Applications on Mac OS X	Room J **Thurs., 9:00am**

Roadmap

511 Games Solutions 1: Graphics, Events, and Tidbits	Room C **Thurs., 10:30am**
512 Games Solutions 2: NetSprocket and OpenPlay	Room C **Thurs., 2:00pm**
513 OpenGL: Advanced 3D	Room J **Thurs., 3:30pm**
514 OpenGL: Performance and Optimization	Room J **Thurs., 5:00pm**

Roadmap

FF012 Core OS Networking:

Room J1
Fri., 2:00pm

Who to Contact

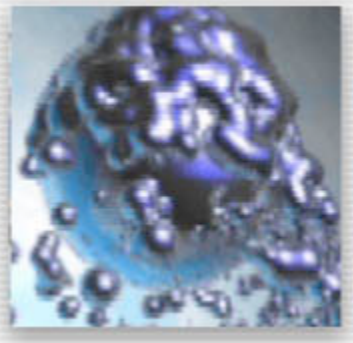
Lane Roathe
President, Ideas From the Deep
lane@ifd.com

Ron Dumont
Darwin Program Manager
rond@apple.com

For More Information

- OpenPlay Home Page
  developer.apple.com/darwin/projects/openplay

Q&A



Lane Roathe
President, Ideas From the Deep
lane@ifd.com

http://developer.apple.com/wwdc2002/urls.html


WWDC2002


WWDC2002


WWDC2002